Effects of Partial Observability Solver Methods on Training and Final Policies in Autonomous Driver RL

How do different methods for dealing with partial observability in the environment influence training and the robustness of final policies under various testing conditions?

Ata Ertuğ Çil[1]

Supervisor(s): Matthijs Spaan[1]**, Moritz Zanger**[1]

[1]EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 25, 2023

Name of the student: Ata Ertuğ Çil, a.e.cil@student.tudelft.nl
Final project course: CSE3000 Research Project
Thesis committee: Matthijs Spaan, Moritz Zanger, Elena Congeduti

Abstract

Autonomous driving is a complex problem that can potentially be solved using artificial intelligence. The complexity stems from the system's need to understand the surroundings and make appropriate decisions. However, there are various challenges in constructing such a sophisticated system. One of the main challenges is to make the agent learn from the environmental input and since the environment is not fully observable and constantly changing, the agent should be flexible enough to extract important information from the input. To create such an agent this paper focused on the 3 different methods, specifically the application of frame stacking, long short-term memory (LSTM), and a combination of both methods. To analyze these methods 3 Deep Q Network(DQN) based agents are created. These 3 agents have frame stacking, LSTM, and both combined methods to solve the partially observable problem of autonomous driving. Their training performances are analyzed, and the results revealed significantly different trends in the training and evaluation phase.

Especially the experiment resulted in LSTM being a more robust method but had lower performance than DQN with frame stack, which showed a trade-off between these 2 qualities of an agent. The agent with LSTM and frame stack was able to learn faster at the beginning, but as it was unstable, it got a lower return value from the training run at the end. This instability can be a problem in the real world, especially in autonomous driving, where it is very important to have robust implementation.

1 Introduction

Autonomous driving is gradually integrating into our lives. Soon, it is expected to be more common, and this raises concerns about the safety of the system[8]. Apart from the mechanical components of the car, the agent is trained to choose an action for the car, and as the agent is deciding instead of the driver what action to take, the system's decision-making process is important. However, this process becomes complex due to the agent's limited ability to understand its surrounding in a continuously changing environment. The cars that are around the agent can be moving or slowing down, but the agent cannot understand this from one frame, and this can create unwanted situations such as crashes or sudden rough actions, which reduces the driver's trust in these agents. To prevent such situations, valuable information regarding the environment should be extracted so that the agent can take action even when faced with changing environment[15]. By reaching these pieces of information and understanding the entire state, the agent overcomes the partial observability problem.

Partial observability problem is as highlighted an important topic in the area that can worsen the decision-making process by introducing uncertainty and potentially leading to suboptimal or unsafe decisions. But this method cannot work standalone, therefore one artificial intelligence algorithm is needed for the agent. As we need experience-driven autonomous learning, Deep Reinforcement Learning especially Deep Q Network (DQN) is chosen as the main algorithm. By having an algorithm that can find low dimension representation of high-dimensional input[19], It is a good choice as a base algorithm as every input consists of many pixels and each of them is a dimension, making the environment has a high-dimensional input.

With the DQN, the 2 methods to solve partial observability are chosen as long short-term memory (LSTM)[12] and frame stacking for 2 agents. While these 2 methods are the most famous methods to solve partial observability, we additionally combined these 2 methods in our third agent. With these, in this paper the question "Partial observability: How do different methods for dealing with partial observability in the environment influence training and the robustness of final policies under various testing conditions?" is answered by comparing these 3 agents that are trained in the CARLA[10] (highly realistic car agent simulator) environment.

The requirement for an answer to this question stems from the safety requirement for the people that can potentially use such systems in the future and the knowledge gap in the literature about the topic currently. For instance, Antonin Raffin compared LSTM combined frame stacking with frame stacking in proximity policy optimization (PPO) based algorithm while examining performance in a nonrealistic car racing simulator[17], which hindered a detailed analysis of each individual method and results that are not applicable to real-world scenarios. Similarly, other sources lack a comprehensive comparison of these methods in an autonomous driving environment. In the paper by Hausknecht and Stone, the methods of frame stacking and LSTM was used in DQN and these different methods have been compared, but the training and testing environment was not a car driving simulation[11]. They mainly focused on Atari games and they did not combine these different methods which is an interesting architecture that could have been investigated. Therefore we think that training agents in a highly realistic environment can lead to different results.

In this paper, we will first give detailed background information regarding the architecture and the techniques that are used in the agents. Subsequently, explain the methodology by giving steps that are required to replicate the research. Following that, experiment setup, initial testing, found results, and discussion with limitations are given. Afterward, responsible research, where critical components in the process are analyzed. Finally, final conclusion with future work is discussed.

2 Background Information

In this section, explanations of important terminologies and techniques that are used in the experiment are given.

2.1 Partial observability

In the real world, it is not common to have a situation where the environment around the agent is fully observable as the agent does not know the state that it is in properly. To solve this, the agent needs to map beliefs to actions where beliefs

are probability distributions over states[20]. This is called the Partially Observable Markov Decision Process (POMDP) which is a generalization of Markov Decision Process (MDP) that is generally used in fully observable situations.

2.1.1 MDP vs POMDP

In the CARLA environment, the information that the agent takes is frames per step, this makes the autonomous driving initially POMDP. However, with the usage of LSTM and frame stacking, we can make the agent able to extract the state that it is in. Therefore the problem of autonomous driving becomes MDP after solving the POMDP with given methods[11].

2.2 Deep Q Network (DQN)

In recent years, there have been many different algorithms created to solve MDP problems[14]. One such algorithm is the Deep Q network. It is a neural network that uses Q learning which returns values for each action that an agent can possibly take. Therefore action space is assumed as discrete.

There are many ways to construct a DQN as there are different kinds of techniques for each component of the algorithm. The chosen techniques to build our DQN agents are Q learning, Epsilon-greedy exploration, and experience replay.

2.2.1 Q learning

Q learning[23] is a model-free, value-based, off-policy algorithm that allows agents to choose the best action that they can take given their state. This means that the agent learns with experiences that are stated as consequences of actions taken by the agent. Additionally, It has a value function that learns which states are more important and takes action regarding this[3].

Q learning allows the agent to store a Q table which consists of sets of actions and states. The values are updated through time and with this Q values start to become similar and accurate for states.

2.2.2 Epsilon-Greedy Exploration

To continuously improve the policy of the agent, exploration, and exploitation need to be in balance. However, there is a trade-off between each other and exploration methods are investigated in the reinforcement learning field for a long time[7].

One of the most common and simple methods is Epsilon-Greedy Exploration which the agent has a uniform chance to take any random action if the chosen random number is lower than the epsilon. This epsilon starts high and decreases in every time step. By decreasing the epsilon the random actions are decreased as time pass and the agent starts to decide more about its actions. This allows the agent to solve the trade-off problem between exploration and exploitation. If the agent was more tended to do exploration even the time passed, it could have lost the learned information, or oppositely if it had more tendency to exploit it could be stuck in a location where it cannot improve its performance without trying new actions.

2.2.3 Experience Replay

It is important to be able to train the agent with the limited data we have. The experience reply method enables this by repeatedly presenting the data to an agent to train without bias. The logic behind this is that, in every time step the experience of the agent is stored in a memory, and the agent randomly samples from this memory as the size of the batch to update its policy. As the data sampled is not sequential, the agent is able to learn from distinct experiences[2].

2.3 Frame Stacking

In the car racing situation, If the car only sees 1 frame as input, how fast the car goes would not be comprehended by the agent, which can create problems when taking turns. For example, if the speed is too high the agent needs to slow before making a turn to slow down. Similar to this example, there is different information that cannot be understood by the agent with only using 1 frame per input. As this makes the situation POMDP, the frame stacking method helps to make the situation MDP which allows us to use our DQN agent.

Frame stacking works as at the beginning after the first frame taken, the input becomes 4 of this same frame, after every step the new observation is put at the end of the stack and returns the last 4 frames. The number of frames can change from situation to situation but as Hausknecht and Stone suggested that 4 frames are enough for the agent to comprehend most of the environmental information[11], our frame stacking is also using the last 4 frames.

These 4 frames are used in Q learning which allows for agents to understand time-dependent information. It is seen that many of the reinforcement learning agents that need to extract information depending on time use this method, which is robust and fast.

2.4 Long Short Term Memory

The second method to make POMDP to MDP is LSTM. Adding this network to DQN makes it a Deep Recurrent Q Network (DRQN). DRQN understands important information that changes through time by using a recurrent layer with the help of the LSTM method. This recurrent layer in the network allows to store internal states and when new input is given the outcome is the result of q values and internal states. This layer has been put at the end of the network as it is shown that the LSTM layer performs best at the end of the network[11].

As the agent has a system that continuously uses internal states, the agent “remembers” or “forgets” parts of the information regarding the environment. The ones that are important in the environments are remembered after the successful episode and updated in the Q table. After each episode, the internal state resets to not create inter-episodic information. To train the DRQN another method called Backpropagation Through Time(BPTT) is needed.

2.4.1 Back Propagation Through Time

BPTT makes agents able to train by using the last n time steps. Similar to frame stack, but in this method after the input of the last n time steps given to the agent, the agent puts them in the LSTM layer n times by consecutively calculating the hidden states and using them for the next step for the cell again n times. Therefore the output of the first layer becomes the input of 2nd layer and so on.

The chosen number for the last n time steps is 4. Even though the LSTM's biggest advantage is being able to learn from a long history, an increased number of time steps can create problems of gradients explosion or vanishing[21]. Also as the number of time steps used affects the training time greatly, the number 4 was found suitable as it gave similar training time as DQN.

2.5 DRQN with Frame Stacking

In the 3rd agent, the DRQN with frame stacking is implemented. This structure is not common in literature as DRQN can choose action even with passing one frame to network[4]. In this architecture, the agent gets 4 last time steps in not only the Q value update phase but also in the action decision process too. This is the change compared with the DRQN as DRQN was only able to see 1 frame while calculating the Q values for each of the actions.

3 Methodology

To successfully run the research and find the proper results, a methodology with clear steps is needed. Therefore the following steps describe the detailed process to conduct the research as well as where and how to compare the findings.

1. Setting up CARLA in Delft Blue[9] (supercomputer of TU Delft) and connecting it with the Gymnasium framework[5]:
    - Collaborate with the team to set up the CARLA simulator in Delft Blue.
    - Establish the connection between CARLA and the Gymnasium framework using the gym-carla[6] third-party environment.
2. Agent Model Implementation:
    - Implement the DQN agent model.
    - Utilize the DQN implementation with frame stacking of 4 as provided in CleanRL[13].
    - Implement another agent with the LSTM, creating a DRQN agent which has backpropagation through time with 4 time steps.
    - Implement another DRQN agent which has backpropagation through time with 4 time steps and has a frame stacking method with 4 frames in each input.
    - Train the agents in the Gym Car racing environment and evaluate their performance for initial testing.
    - Adjust the training parameters as needed for optimal performance.
3. Data collection and evaluation:
    - Create new environments in Carla and train agents with the usage of the Gym-Carla wrapper.
    - Evaluate the agents in 3 different environments in Carla, each having 10 episodes per environment.
    - Use TensorFlow[1] to create graphs that visualize the results of the three agents over time (time steps) for training.
    - Use Excel[16] to create a graph that shows agents' mean episodic return for each environment.
    - Find trends between agents' training phase and compare their performances. Find which agent is more robust and effective.

By following this methodology, CARLA is set up in Delft Blue and integrated with the Gymnasium framework. The agent models are implemented and trained by using DQN with LSTM, frame stacking methods, and combined. The agents' training performance, final policies, as well as evaluation of their robustness in environments are compared.

4 Experiments

In this section, the experimental setup is explained more in detail, gave the initial experimental results and the results of the training and evaluation phases of agents in Carla environment are discussed.

4.1 Experimental setup

For the initial tests in the Gymnasium environment, some decided packages are used within the Conda environment. The list of modules with versions can be seen in Appendix A. In this environment, to build up an agent that uses DQN with frame stacking, an agent that was implemented in CleanRL[13] repository is taken. This chosen agent was dqn_atari, as the other agent called dqn in the repository was not suitable for input from the environment. dqn_atari agent had frame stacking implemented which uses the last 4 frames, therefore using the dqn_atari code, an agent that uses DQN with frame stacking in the car race environment of the Gymnasium is built.

To build the second agent that uses DRQN, the network architecture in the paper[11] from Hausknecht and Stone is used. The base DQN agent that uses frame stacking is copied and changes are done to translate the agent into DRQN. One of the main changes was adding an LSTM at the end of the convolutional layers as the paper supported that this architecture showed greater performance when compared to other architectures. The other change is that in the action decision phase where the agent estimates q values for each action with a given input, the agent was only able to see one frame, not 4 frames. As the agent's hidden states are carried over the episode, giving 4 frames as input to the agent was not required and wanted. Apart from that BPTT is implemented in the training phase which the agent uses the last 4 time steps to train.

The third agent uses DRQN with frame stacking. To implement this, DRQN which looks at only the last frame to choose an action changed that it can look at 4 frames, and as the BPTT was also 4, the training phase looks at the last 4 frames too. The recurrent layer keeps the network still as DRQN.

After these agents are implemented, they need to be able to run in the Gymnasium environment and Carla after. To be able to run in Carla the Gymnasium needs to be connected to Carla. This is done by using a gym-carla wrapper. To make Gymnasium and Carla compatible with the gym-carla

Parameter name	Value
start-e	1
end-e	0.05
learning-rate	2.50E-04
exploration-fraction	0.5
learning-starts	10000
train-frequency	10
buffer-size	50000
gamma	0.99
tau	1
target-network-frequency	500
batch-size	128

Figure 1: Agents parameter values

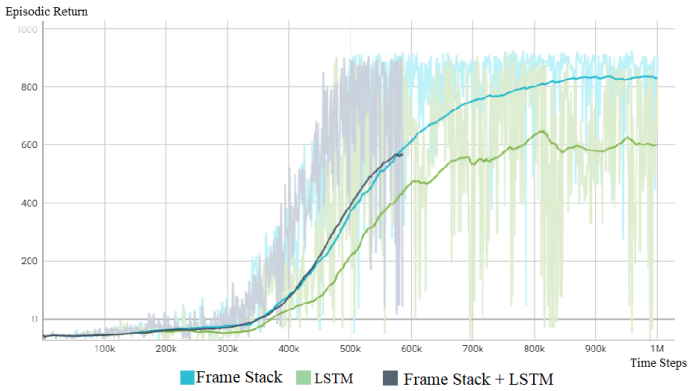

Figure 2: smoothed episodic returns of 3 agents that are trained in Gymnasium environment

wrapper, Gymnasium version 0.26.2 and Carla version 0.9.13 are used.

In both environments, the parameters in Figure 1 remained the same for the training phase for all agents. The variable total time step was 1 million in the Gymnasium environment and 500 thousand in the Carla environment. All of the parameters used for Carla Environment can be seen in Appendix B. Additionally, 3 different tracks are used during the evaluation phase in the Carla simulator to be able to highlight the robustness of the methods.

It is important to know that in the CARLA environment, the agent gets rewarded by a weighted combination of longitudinal speed and penalties for collision, exceeding maximum speed, and large lateral acceleration. while termination conditions are the ego vehicle colliding, going out of lane, reaching a destination, or reaching the maximum episode time steps. These are used in the discussion of the results.

4.2 Initial Experimental work

Before starting to train agents in Carla, a less realistic environment which is Gymnasium simulator is used. This step allowed us to see initial trends and observe the implemented methods' impacts.

The smoothed episodic return in the Gymnasium environment of each agent's trends is given in Figure 2. In the training, the DQN agent that stacks 4 frames the episodic return changed to a positive direction at around 400k time step which means after close to 400 runs of Car driving simulation in the Gymnasium environment the agent started to learn efficiently. At 1 million time step the agent was getting an average of close to 830 episodic returns which shows that it was successful in training in the tracks of the Gymnasium. The agent could train more but it would not have better performance as it has not improved performance significantly in the last 300k time steps. For the initial testing, the agent's trend showed enough evidence to continue the experiment. After testing DQN with the frame stacking, the DRQN agent with BPTT=4 is trained. As seen in Figure 1, it had started to train later than DQN but was able to stay in a consistent average episodic reward. It is seen that the final average episodic return at 1 million time step was close to 600 which showed that it performed not as well as DQN. This difference could have happened because of the chosen number of n for BPTT, or it can also be a case where the car race environment is not suitable to DRQN as much as DQN with frame stacking. Even though it performed worse it is seen that the DRQN was able to learn and go on the car race track with only receiving one frame per step. As LSTM layer in the network of DRQN is the only layer that can store information regarding the past, it is seen that LSTM was able to store important information and help the DRQN agent to detect features like the agent's own velocity to turn the corners properly.

Lastly, the training performance of DRQN with frame stacking can be seen in Figure 2. This agent was able to start learning before the time step that DRQN did however the steps per second were lower than DRQN, which resulted in reaching 584k steps after 1 and a half days of training. This showed that this agent required more sources to train than DQN as DQN with frame stacking only required around 11 hours of training and for the sake of the experiment, the positive trend was enough to continue the experiment in Carla environment to see the performances in a more realistic environment.

After these tests, the initial thoughts on the agents started to build up, and continued to experiment by setting up Delft Blue and connecting to CARLA with the usage of gym-carla.

4.3 Training Phase

After trained and having initial thoughts on the agents in Gymnasium environment, we made them able to train in the CARLA environment by changing some lines of code as specified in gym-carla wrapper. After creating proper scripts and folders for agents, the agents are trained in Delft Blue for around 20-23 hours at the end making 500 thousand total time steps. When the training is finished, the logs of each agent's training phase are combined to create Figure 3 in TensorFlow.

When the early time steps were analyzed, the result was not expected as the training in the Gymnasium environment supported that DQN with frame stacking would start to learn faster than DRQN while DRQN with frame stacking would perform similarly. However, in Figure 3 it is seen that DRQN with frame stacking was very early to start learning, reaching the point that can be counted as a good episodic return at 130k time step where DQN reached there at 300k time step. This point can be called a good episodic return value as DQN finishes the training at that point. This difference between the starting time steps of learning can be con-

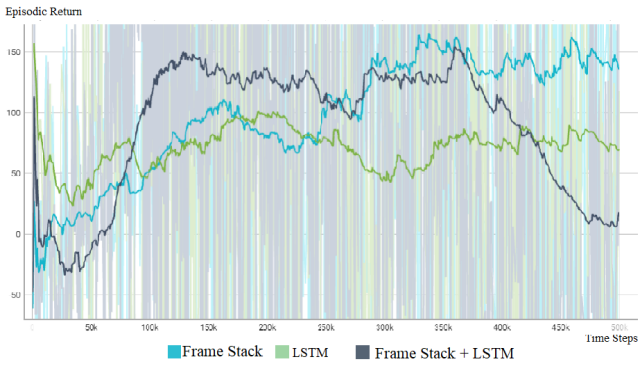

Figure 3: Smoothed Episodic returns in the training process of agents in Carla

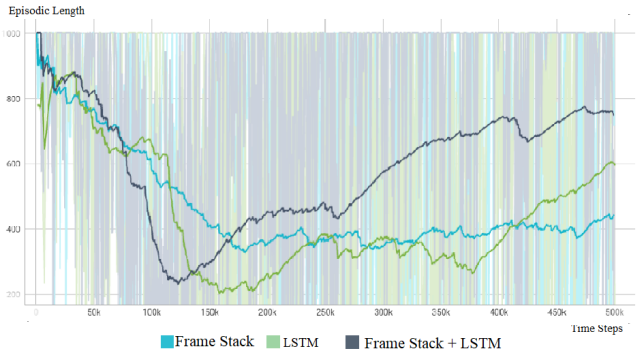

Figure 4: Smoothed Episodic lengths in the training process of agents in Carla

nected to CARLA being a more realistic environment with other cars and pedestrians passing, and the importance of the quality of the method used to solve partial observability increases. DRQN with frame stacking was possibly extracting more information regarding the environment which reduced the collision possibility and returned reward more. To support this, Figure 4 can be used. It is seen that when DRQN with frame stack reached episodic return peak at 130k time step, the episodic length of it was at an all-time low. DRQN with frame stack's episodic length decreased faster than other agents which supports that it was more efficient in navigating in the environment and completing required tasks. Also when looking at Figures 5 and 6, it is seen that there are no big differences between agents regarding the difference between their expected return and actual return (TD loss), while DQN overestimates the Q value. DRQN and DRQN with frame stacking had lower Q values in these early steps which necessarily does not show a disadvantage but rather actions returning value is not as high as it is in DQN with frame stack possibly as the state is observed differently in these. Additionally, the similar TD loss supports that all agents were similarly efficient at updating their Q values.

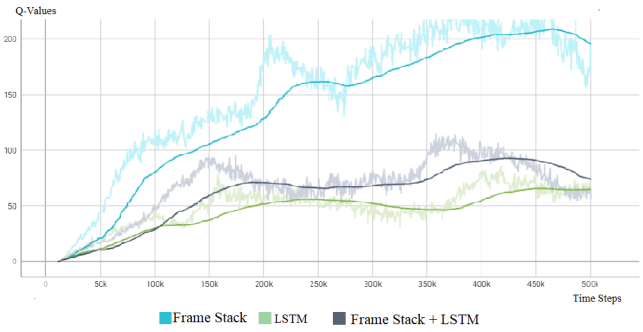

Figure 5: Smoothed Q Values in the training process of agents in Carla

Another unexpected trend was the DRQN with frame stacking's episodic return between 360k to 500k time steps. DRQN with frame stacks starts to have a negative trend in episodic return at 360k time step, reaching closer to value 18 at the end of the 500k time step. This event can be explained by having an unstable agent that tries to learn but forgets the learned. This can also be called catastrophic interference or catastrophic forgetting. this can happen when Q function cannot decide on the optimal choice as there is a state that is similar to learned before but the same action results in punishment. It is also seen in Figure 5 that DRQN actually did not estimate the return too high and the loss between the predicted and actual was very low while in Figure 5 that is visible that the episode length increased for DRQN with frame stacking at the end steply. This suggests that the agent starts to stay still more than moving around. With more training time we think that it could be fixed and return to its older values in Figure 3.

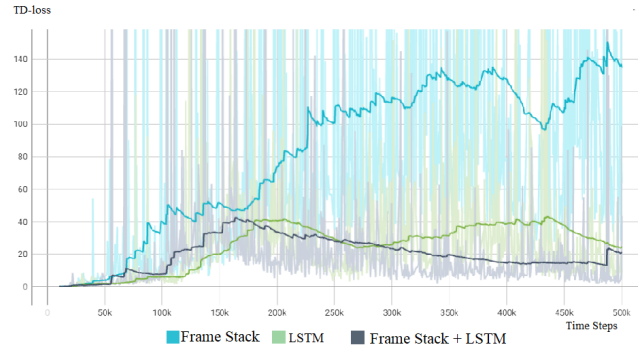

Figure 6: Smoothed temporal difference (TD) loss in the training process of agents in Carla

Lastly, the low episodic return of DRQN was expected, as in the Gymnasium environment DRQN get an episodic return

of 600 while DQN with frame stacking gets 800. The difference was 200/800 which is %25 while the difference in Carla environment is around 140 to 90 making %35 difference. As explained in initial experiments with the Gymnasium environment, the car racing can be not as suitable to DRQN as it is to DQN with frame stacking. The bigger gap between the agents can be happened because of the Carla's being more realistic, making DRQN less suitable for car simulation. It is also seen that the TD loss was very low at the end while Q values were also lower than DQN with frame stacking, therefore more than 500k timestep would be beneficial to see any future trends between the agents as the performances can vary. Training the agent more would be able to make the agent updates its q values better as at this point it is not as efficient as DQN with frame stacking.

4.4 Evaluation Phase

After training the agents in the Carla environment, we run the agents in each town for 10 episodes to compare the robustness and the performance of the final policies. The results of these runs are given in Figures 7 and 8.

It is visible that DQN with frame stack outperformed other agents in every environment and had a greater average episodic return in Town04 especially. As Town04 is an infinity-shaped road with no limit of the end, DQN with frame stack was able to stay more alive than the other agents in some episodes while having a great standard deviation which shows a high variability in result, meaning that it is not robust as the expected performance is mostly not stable. When DRQN is

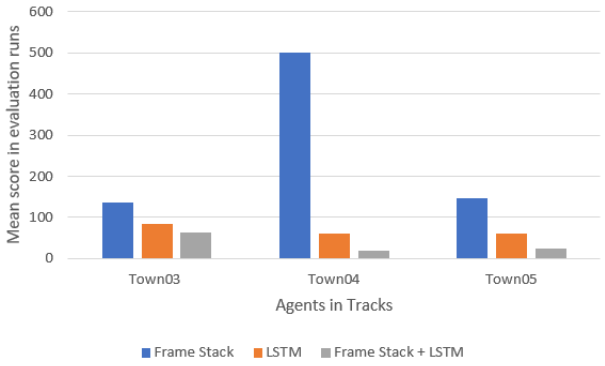

Figure 7: Mean episodic return of 3 agents 10 episodes in 3 different tracks

investigated it is seen that it performed as expected from the training data and its standard deviation is less than DQN with a frame stack. This shows a trade-off between performance and robustness as DRQN is more robust while having less good performance.

In the case of DRQN with a frame stack, it performed worse in every environment. This was expected as DRQN with frame stack had experienced catastrophic forgetting which resulted in losing past information and skills that were learned before. This suggests that DRQN with frame stack has issues with extracting information and using this information in new environments. In the case of robustness,

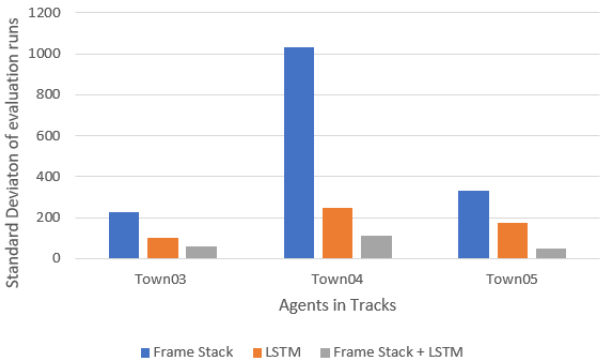

Figure 8: Standard deviation of 3 agents 10 episodes in 3 different tracks

DRQN with frame stack agent's overall robustness is considered compromised despite having a low standard deviation.

Discussion

After the experiment, the following results are found. DQN with frame stacking was the fastest in the training phase but started to learn at a later time step than other agents. Its performance in the evaluation phase is best among agents however it showed the highest standard deviation therefore it is also less robust than other agents.

DRQN agent had lower performance and slower training times than DQN with frame stack which can be explained by the complexity that the LSTM layer adds to the network. In the evaluation phase, DRQN performed as expected from the training data. While it performed low, its standard deviation was lower, making it more robust than DQN with a frame stack. The difference between the performance is also seen in the paper of Hausknecht and Stone, where only 5 out of 9 games DRQN outperformed the DQN with frame stacking[11]. In 2 of the games, the DRQN actually performed less than half of than DQN with frame stack performed, which is similar to the situation in our experiment. Additionally in the paper of Romac and Béraud, they compared the DQN and DRQN in the Minecraft (3d game) environment, where DRQN has performed lower than DQN with frame stack again [18]. This shows that DRQN is not as suitable for autonomous driving as much as DQN with frame stack, but if one wants robustness over the performance then DRQN is a more viable option.

The DRQN with frame stack is not an architecture that was investigated in any research before, therefore the results are new for the field. The result actually showed that it started to learn in earlier time steps than other agents and performed as well as DQN with frame stack until it experienced catastrophic forgetting. This caused us to consider its robustness compromised as the agent become like it did not train at all, remaining more idle than moving. This makes it an undesirable agent and needs to be investigated more.

Limitations

Running a training script in CARLA for 1 million time steps was taking more than 1 day which is not allowed by Delft

Blue. In 24 hours, agents were able to train 500k time steps and to be able to train them, checkpoints could have been used. However, as Delft Blue was busy with many different jobs, each experiment run was delayed. This was a big problem as to test if the agents' code was working, we needed to schedule jobs in Delft Blue, but as it was very busy, the starting times for jobs could find 2 days. When added the training time of 1 day, it was time-consuming to test training and redo the process in a time of 9 weeks. The DRQN with frame stacking would be compared with other agents better by solving this limitation, however, the catastrophic forgetting has also given a good insight into how unstable the agent is.

5 Responsible Research

From the responsible perspective of the research, it is important to eliminate the possibility of getting different results when re-conducting the experiment.

Firstly, the Neural Networks inherit randomness as a basis. This randomness can easily affect the outcome of the result. To prevent this a specific seed is used throughout the experiment which was 0. This seed was selected after trying out different seeds and finding the ones that did not give positive or negative bias to agents. Additionally to reach closer results the number of trials in this research could be increased which would help to get closer results when wanted to reconduct.

Secondly, apart from DQN with frame stack implementation, the DRQN and DRQN with frame stack agents are highly dependent on the architecture chosen by us as the DQN with frame stack algorithm is very similar to the one in CleanRL. This dependency on the architecture can affect the process. Also, it is important to state that the results of the DQN with frame stack in the training and evaluation phases are shared among the researchers while conducting the experiment. The reason behind this was using the same algorithm with the same parameters in the same environment with the same seed more than 1 time would not benefit the research, therefore to fasten the experiment the base algorithm results are used to compare in the analysis part.

Lastly, when replicating the experiments it is important to care that, the results cannot be used in real-world situations without thorough investigation and being sure. As autonomous driving is a sensitive topic in the real world, it is needed to be careful.

6 Conclusions and Future Work

In this research, our purpose was to answer the question "How do different methods for dealing with partial observability in the environment influence training and the robustness of final policies under various testing conditions?". The methods investigated were frame stacking, LSTM, and these 2 combined. These methods were implemented into DQN agents and the created agents' training and evaluation phases are analyzed. From the analysis of the training phase, it has been seen that the DRQN with frame stacking had started to learn earlier than other agents while DQN with frame stacking has finished the training in the shortest time and highest episodic return. DRQN finished the training with lower episodic return than DQN with frame stack while DRQN with frame stack has experienced catastrophic forgetting. This showed that an additional LSTM layer in the network increased the complexity and slowed down the training speed. In the evaluation phase, expected findings were observed where DQN with frame stack had the highest average episodic return and DRQN with frame stack scored lowest. Additionally, new findings are found regarding the robustness of these agents. While DRQN had a lower average episodic return its standard deviation was lower than DQN with frame stacking, making it more robust than DQN with frame stacking in the domain of autonomous driving. To be able to get better results in the future, agents could have taken more time steps in the training phase until they converge. This would allow DRQN with frame stacking to overcome its problem of forgetting. Also more evaluation runs in different runs can be run for every agent.

Additionally, how the parameters used in frame stack and backpropagation through time change the training and evaluation results can be investigated to find different trends in this domain. While DRQN did not increase the performance, this can be due to the usage of BPTT=4, and usage of a higher number for this value can make the agent comprehend the environment better.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Sander Adam, Lucian Busoniu, and Robert Babuska. Experience replay for real-time reinforcement learning control. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(2):201–212, 2012.

[3] Abid Ali Awan. An introduction to q-learning: A tutorial for beginners, Oct 2022.

[4] Madhuparna Bhowmik. Deep recurrent q-network, Jan 2019.

[5] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[6] Jianyu Chen. gym-carla. https://github.com/cjy1992/gym-carla, 2020.

[7] Will Dabney, Georg Ostrovski, and André Barreto. Temporally-extended epsilon-greedy exploration, 2020.

[8] Johannes Deichmann, Eike Ebel, Kersten Heineke, Ruth Heuss, Martin Kellner, and Fabian Steiner. Autonomous driving's future: Convenient and connected, Jan 2023.

[9] Delft High Performance Computing Centre (DHPC). DelftBlue Supercomputer (Phase 1). https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase1, 2022.

[10] Alexey Dosovitskiy, Germán Ros, Felipe Codevilla, Antonio M. López, and Vladlen Koltun. CARLA: an open urban driving simulator.

[11] Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps, 2017.

[12] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[13] Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João G.M. Araújo. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23(274):1–18, 2022.

[14] Alex Kendall, Jeffrey Hawke, David Janz, Przemyslaw Mazur, Daniele Reda, John-Mark Allen, Vinh-Dieu Lam, Alex Bewley, and Amar Shah. Learning to drive in a day, 2018.

[15] Ugo Lecerf, Christelle Yemdji-Tchassi, and Pietro Michiardi. Safer autonomous driving in a stochastic, partially-observable environment by hierarchical contingency planning, 2022.

[16] Microsoft Corporation. Microsoft excel.

[17] Antonin RAFFIN. Ppo vs recurrentppo (aka ppo lstm) on environments with masked velocity (sb3 contrib), May 2022.

[18] Clément Romac and Vincent Béraud. Deep recurrent q-learning vs deep q-learning on a simple partially observable markov decision process with minecraft, 2019.

[19] Ahmad EL Sallab, Mohammed Abdou, Etienne Perot, and Senthil Yogamani. Deep reinforcement learning framework for autonomous driving. *Electronic Imaging*, 29(19):70–76, jan 2017.

[20] Matthijs T. J. Spaan. Partially observable Markov decision processes. In Marco Wiering and Martijn van Otterlo, editors, *Reinforcement Learning: State of the Art*, pages 387–414. Springer Verlag, 2012.

[21] Ralf C. Staudemeyer and Eric Rothstein Morris. Understanding lstm – a tutorial into long short-term memory recurrent neural networks, 2019.

[22] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[23] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992.

A Package List in Gymnasium Environment

- absl-py=1.4.0
- box2d-py=2.3.5
- ca-certificates=2023.01.10
- cachetools=5.3.0
- carla=0.9.13
- certifi=2023.5.7
- charset-normalizer=3.1.0
- cloudpickle=2.2.1
- colorama=0.4.6
- contourpy=1.0.7
- cycler=0.11.0
- decorator=4.4.2
- filelock=3.12.0
- fonttools=4.39.4
- google-auth=2.18.0
- google-auth-oauthlib=0.4.6
- grpcio=1.54.0
- gym=0.26.2
- gym-notices=0.0.8
- idna=3.4
- imageio=2.28.1
- imageio-ffmpeg=0.4.8
- importlib-metadata=6.6.0
- importlib-resources=5.12.0
- jinja2=3.1.2
- kiwisolver=1.4.4
- lz4=4.3.2
- markdown=3.4.3
- markupsafe=2.1.2
- matplotlib=3.7.1
- moviepy=1.0.3
- mpmath=1.3.0
- networkx=3.1
- numpy=1.24.3
- oauthlib=3.2.2
- opencv-python=4.7.0.72
- openssl=1.1.1t
- packaging=23.1
- pandas=2.0.1
- pfrl=0.3.0
- pillow=9.5.0
- pip=23.0.1
- proglog=0.1.10
- protobuf=3.19.6
- pyasn1=0.5.0
- pyasn1-modules=0.3.0
- pygame=2.1.0
- pyparsing=3.0.9
- python=3.8.2
- python-dateutil=2.8.2
- pytz=2023.3
- pywavelets=1.4.1
- requests=2.30.0
- requests-oauthlib=1.3.1
- rsa=4.9
- scikit-image=0.16.2
- scipy=1.10.1
- setuptools=66.0.0
- six=1.16.0
- sqlite=3.41.2
- stable-baselines3=1.2.0
- swig=4.1.1
- sympy=1.12
- tensorboard=2.10.0
- tensorboard-data-server=0.6.1
- tensorboard-plugin-wit=1.8.1
- torch=2.0.1
- tqdm=4.65.0
- typing-extensions=4.5.0
- tzdata=2023.3
- urllib3=1.26.15
- vc=14.2
- vs2015_runtime=14.27.29016
- werkzeug=2.3.4
- wheel=0.38.4
- Use latex to have consistent formatting amongst teammates. The *ijcai* template provided is should be the standard.

B Carla Parameters

'number_of_vehicles': 100,

'number_of_walkers': 0,

'display_size': 256, # screen size of bird-eye render

'max_past_step': 1, # the number of past steps to draw

'dt': 0.1, # time interval between two frames

'discrete': False # whether to use discrete control space

'discrete_acc': [-3.0, 0.0, 3.0], # discrete value of accelerations

'discrete_steer': [-0.2, 0.0, 0.2], # discrete value of steering angles

'continuous_accel_range': [-3.0, 3.0], # continuous acceleration range

'continuous_steer_range': [-0.3, 0.3], # continuous steering angle range

'ego_vehicle_filter': 'vehicle.lincoln*', # filter for defining ego vehicle

'port': 2000, # connection port

'town': 'Town03', # which town to simulate

'task_mode': 'random', # mode of the task, [random, roundabout (only for Town03)]

'max_time_episode': 1000, # maximum timesteps per episode

'max_waypt': 12, # maximum number of waypoints

'obs_range': 32, # observation range (meter)

'lidar_bin': 0.125, # bin size of lidar sensor (meter)

'd_behind': 12, # distance behind the ego vehicle (meter)

'out_lane_thres': 2.0, # threshold for out of lane

'desired_speed': 8, # desired speed (m/s)

'max_ego_spawn_times': 200, # maximum times to spawn ego vehicle

'display_route': True, # whether to render the desired route

'pixor_size': 64, # size of the pixor labels

'pixor': False, # whether to output PIXOR observation